UNIVERSITY OF
CAMBRIDGE

Comparing Machine Learning Techniques for Mobility Graph Classification

Miteyan Patel

Robinson College

Submitted on 20th May, 2018

Abstract

Declaration

I, Miteyan Patel of Robinson College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed:

Date:

Miteyan Patel

December, 2017

Acknowledgements

Contents

Chapter 1

Introduction

Machine learning techniques typically use structured and organised data which may not always be easily available in every day life. I will be investigating the use of unstructured graphical data as input into the most popular machine learning algorithms to evaluate the practicality of this and also apply new cutting edge algorithms to determine if we can classify users based on graphs created from their smartphone location data to infer their demographic class as labels as a supervised learning problem. Then I will create a mobile application to collect then cluster the location data into graphs for use by the most successful of the supervised algorithms investigated. If successful, this can have the great implications towards the utilisation of the data we have and can use in the future to create better classifiers and for example this can be used as a framework into using location data to infer whether or not users have neurological diseases such as Alzheimer's which have known to alter the movement patterns of its carriers.

Chapter 2

Preparation

My preparation consisted of background reading and thoroughly understanding various supervised machine learning algorithms so that I could optimize them, as well as studying common techniques in conducting a machine learning project such as evaluation of the algorithms. Also researching graph theory and finding out features of graphs that could be extracted from the locations of the users to be used an input into the supervised learning algorithms. This chapter will describe the work which was undertaken before code was written and theories worked on. It will also show how the project proposal was further refined and clarified, so that the Implementation stage could go smoothly rather than by trial and error.

2.1 Supervised machine learning

Supervised learning algorithms are used to infer the mapping function from labeled training data. Which consists of feature vectors and an associated label. Input feature vectors $\mathbf{x} \in \mathbb{R}^m$ are associated with a label y. In classification problems, y represents one of the possible output classes $\mathbf{C} = \{C_1, ..., C_k\}$. A training set is composed of n such training examples:

$$\mathbf{s} = [(\mathbf{x_1}, y_1)(\mathbf{x_2}, y_2)...(\mathbf{x_n}, y_n)]$$

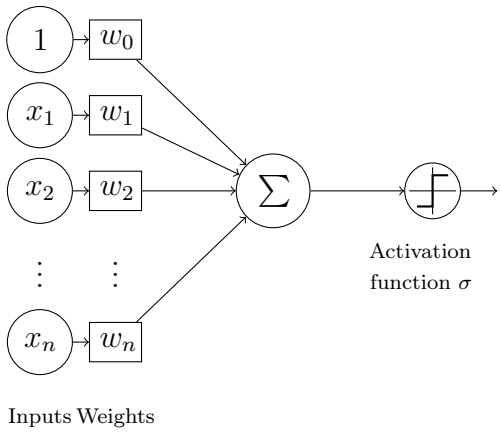
The goal is to approximate the mapping function called the hypothesis $h : \mathbb{R}^m \rightarrow \mathbf{C}$, so well that when you have new input data $\mathbf{x}$ that you can predict the output class for that data.

2.1.1 Neural Networks

Based on the extravagantly complex neuron networks in the brain, a type of supervised learning algorithm commonly used is the Artificial Neural Network. Although they are an oversimplification of the biological neural networks they offer a very successful technique

for regression and classification problems. It defines some function $f(\mathbf{x}; \mathbf{w}, \mathbf{b})$, dependent on its architecture to approximate the hypothesis, where $\mathbf{x}$ is the input vector, and $\mathbf{w}$ and $\mathbf{b}$ are the weights and biases in the network respectively. The Neural Network learns over successive iterations of the training algorithm the values of weights and biases by minimising a loss function, which defines to measure the model's error when estimating y from $\mathbf{x}$, given initially random choices for $\mathbf{w}$ and $\mathbf{b}$.

Perceptron



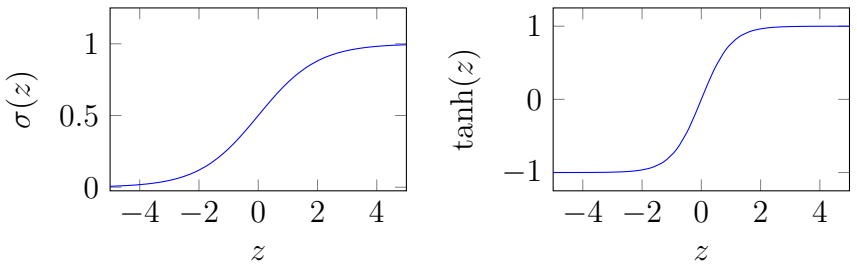
The output of the perceptron is the cross product of the weights and the inputs into it followed by a pass through the activation function, which is used to normalise the output from the perceptron.

The constant-valued input 1 for x_0 is known as a bias input, which is weighted by a bias weight variable, w_0. This allows the output of the network to be shifted left or right to maximise the accuracy and range of values that can be modeled by the perceptron.

$$h(w; x) = \sigma(\sum_{i=0}^{n} w_i x_i) = \sigma(\mathbf{w^T x})$$

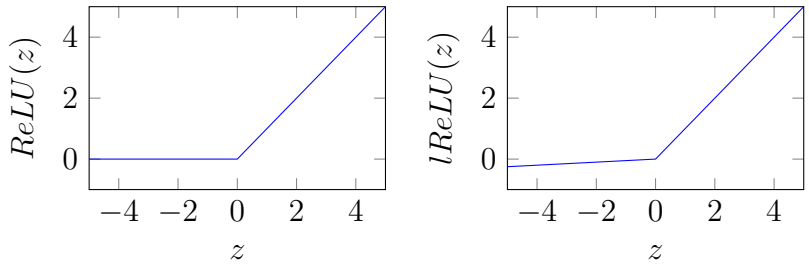
Activation Functions

Activation functions decide whether or not a neuron should be activated or transmit information. The activation function needs to be non-linear which can be proved to be a universal function approximator for a two-layered network and allows for errors to be backpropogated more easily[1] Rectified Linear Units (ReLUs) are used in Convolution Neural Networks, which are in practice more effective than the sigmoid or tanh functions. However at the negative side of the graph, the gradient is zero, which means for activations in that region, the gradient is zero and the weights are not updated during back propagation. This can create dead neurons which never get activated. The leaky ReLU solves this by replacing the horizontal section by a line with a small gradient so there are no more dead neurons for better classification.



(a) Logistic sigmoid

(b) Hyperbolic tangent



(c) ReLU

(d) Leaky ReLU

Figure 2.1: Commonly used activation functions including the logistic sigmoid $\sigma(z)$, the hyperbolic tangent $tanh(z)$, ReLU and leaky ReLU.

Multi-layered Perceptron Perceptrons on their own are not powerful enough to generalise to more complex probability distributions. A multilayer perceptron is a collection of fully connected perceptrons, consisting of at least three layers of non-linear activations. Every perceptron is connected with each in the layer in front with a weight $w_{i,j}$. Learning occurs in the perceptron by changing connection weights after each training data is processed, based on the amount of error in the output compared to the expected result, through backpropogation.

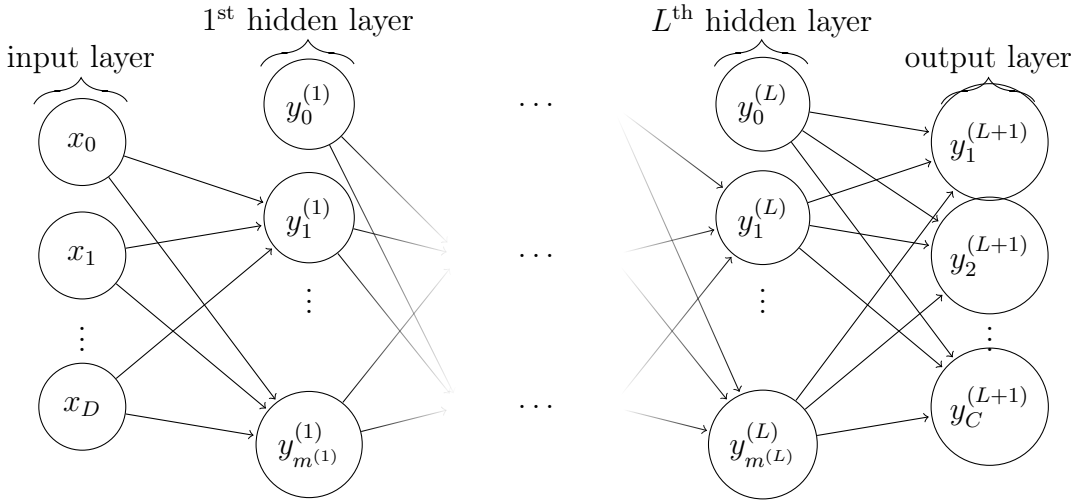


Figure 2.2: Network graph of a $(L+1)$-layer perceptron with D input units and C output units. The l^{th} hidden layer contains $m^{(l)}$ hidden units.

2.2 Convolution Neural Networks

Convolution Neural Networks consist of input, hidden and output layers. The hidden layers unlike normal deep neural networks contain convolutional, pooling, fully connected and normalization layers.

Convolutional

The convolutional layer applies a convolution operator to the incoming neuron layer and

Pooling

Pooling layers are used to reduce the size of the layers. They can take multiple neurons from one layer and output a single reduced neuron. For example max pooling is where the maximum value of the cluster of neurons is used as the output, or average pooling where the mean average of the neuron cluster is used.

Fully Connected

A fully connected layer connects every neuron from the input layer to every neuron in the output layer.

2.2.1 Naive Bayes Classifier

Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with naive independence assumptions between the features. This is the major problem with this approach since the features derived from the graphs are likely dependent on each other, although an advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification. The Bayes theorem states describes the likelihood of an event A given an event B is true probability, based on prior knowledge of conditions that might be related to the event.

$$P(C_k \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid C_k)\, P(C_k)}{P(\mathbf{x})}$$

In reality only the numerator is of interest since the denominator does not depend on the class **C** so it is effectively a constant and can be taken out and rewritten as:

$$\begin{aligned} P(C_k, x_1, x_2, ..., x_n) &= P(x_1 \mid C_k, x_2, ..., x_n) P(x_2, ..., x_n, C_k) \\ &= P(x_1 \mid C_k, x_2, ..., x_n) P(x_2 \mid x_3, ..., x_n, C_k) P(x_3, ..., x_n, C_k) \\ &= ... \\ &= P(x_1 \mid C_k, x_2, ..., x_n) ... P(x_n - 1 \mid x_n, C_k) P(x_n \mid x_n, C_k) P(C_k) \end{aligned} \tag{2.1}$$

Now with the naive assumption that each feature x_i is conditionally independent of one another so:

$$P(x_i \mid x_i + 1, ..., x_n, C_k) = P(x_i \mid C_k) \tag{2.2}$$

Therefore the conditional probability of the class variable C is:

$$P(C_k, x_1, x_2, ..., x_n) \propto P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k) \tag{2.3}$$

Finally to construct the naive Bayes classifier the most probable hypothesis is chosen known as the maximum a posteriori decision rule and the corresponding Bayes classifier is the function that assigns a class label $\hat{y} = C_k$, for some class C_k as:

$$\hat{y} = \underset{k}{\operatorname{argmax}} P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k) \tag{2.4}$$

2.2.2 Support Vector Machines

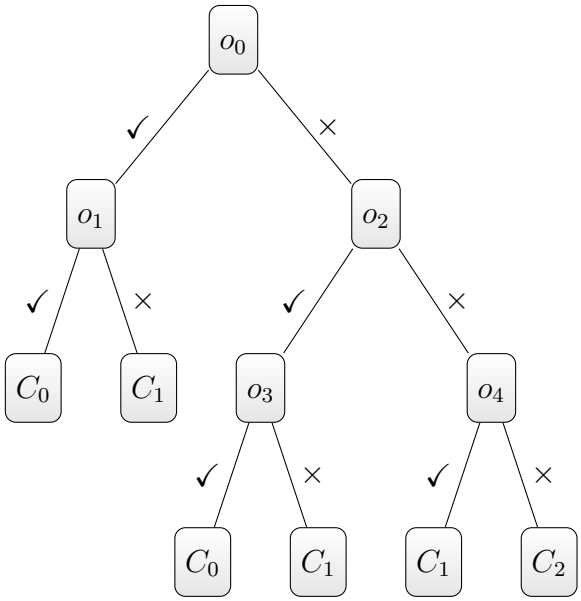
Support Vector Machines a represent the training examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. When new examples are observed they are mapped into that same space and predicted to belong to a category based on which side of the gap they fall. SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

2.2.3 Decision trees

The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. It allows the model to be visualised so the results can be more easily reasoned about compared to other machine learning algorithms which can seem like a black box between the inputs and outputs. Although the trees are prone to overfit to data, particularly if the tree is deep.

A rooted tree is constructed with each branch containing observations about the item of data. The observations are used to traverse down the tree until the conclusion about the data is retrieved at the leaf nodes of the tree.

Figure 2.3: Example of a decision tree where o_i are observations about the data and C_i are classes the datum could belong too once traversing the decision tree from the root observation going left or right depending on the outcome of the observation test.



2.2.4 Random Forests

Random forests are comprised of a multitude of differently constructed decision trees at training time and output the class that is the mode of all the decision tree outputs. Random forests correct the overfitting habit experienced by decision trees[**?**]. They prevent overfitting by constructing many different decision trees using only certain subset of the features vector. Overall, although each tree will only incorporate a small random subset of features, collectively most or all of the features will be utilized by all of the trees. This randomness prevents highly correlated trees since a few features that are particularly predictive could otherwise be used to construct most of the trees with a small training error but so the model would be overfit to this.

2.3 Graph Theory

To use graphs constructed from the user location data as inputs into the traditional supervised machine learning algorithms, the features belonging to the graphs must be extracted from the graphs constructed. I will explain some of the graph features we can extract out to distinguish between graphs that were used in supervised approaches.

Number of nodes: the number of nodes in the graph

Number of edges: the number of edges in the graph

Max degree: the maximum number of neighbours any node in the graph has

Density: Density is defined as:

$$\frac{|E|}{|V|(|V|-1)}$$

for directed graphs, a dense graph is a graph in which the number of edges is close to the maximal number of edges.

Eccentricity: Eccentricity of a node is the maximum distance to any other node in the graph. This is a vector as it is defined for each node in the graph and so the mean and variance of the Eccentricity are used as scalar value.

Diameter: The greatest distance between any two nodes in the graph also the largest eccentricity in the graph.

Radius: The minimum of the greatest distance between any nodes in the graph also the smallest eccentricity value from any node in the graph.

Centre Size: The size of the centre of a graph. The centre of a graph is the set of all nodes in the graph which have minimum eccentricity or are the radius of the graph.

Average Shortest Path Length: The mean average of the shortest path between all pairs of nodes in the graph.

Clustering Coefficient: measure of the degree to which nodes in a graph tend to cluster together.

Betweenness Centrality: The betweenness centrality for each vertex is the number of shortest paths that pass through the vertex. The shortest path between the vertices's is one where the sum of the weights of the edges is minimized.

Shortest Path Length: Average and variance of the shortest path lengths in the graph.

Edge Betweenness Centrality: Vector containing the fraction of shortest paths each edge belongs to. The mean and variance of these are extracted as features.

PageRank: Computes a ranking of the nodes in the graph based on the structure of the incoming links.

2.4 Software Engineering

This section deals with the requirements set out, early decisions made and tools and software engineering techniques used to finish and deliver the project in an organised, timely fashion.

2.4.1 Requirements

2.4.2 Tools and Libraries

Version Control

I used Git for version control for the project, which was put on my GitHub repository.

Build Tools

I used PyCharm for building and testing the Python code since it contains a great SSH server interface for accessing the dataset stored on its private server and for creating and running Python scripts on that server via my local machine, abstracting away complexities in accessing the private, sensitive information to prevent any leaks of data out of its private server.

Languages

2.4.3 Starting Point

Chapter 3

Implementation

Chapter 4

Evaluation

Chapter 5

Conclusion

Bibliography

[1] Cybenko, G.V. Approximation by Superpositions of a Sigmoidal function. In van Schuppen, Jan H. Mathematics of Control, Signals, and Systems. Springer International. pp. 303–314. 2006

[2]